Application Note

A Webpage Interface for the eZ80F91 MCU using Java

AN017801-0304

ZiLOG Worldwide Headquarters • 532 Race Street • San Jose, CA 95126
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
532 Race Street
San Jose, CA 95126
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

ZiLOG is a registered trademark of ZiLOG Inc. in the United States and in other countries. All other products and/or service names mentioned herein may be trademarks of the companies with which they are associated.

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

©2004 by ZiLOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZiLOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZiLOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.

Table of Contents

List of Figures iv
List of Tables iv
Abstract 1
ZTP Overview 1
HTTP Server in ZTP 2
Java Applets and HTML 3
Developing the Java-based Webpage Interface Application 3
Creating the Webpage 6
Controlling the LEDs on the Development Board 15
Controlling the LED Icons on the Webpage 16
Adding and Integrating Application Specific Files to ZTP 16
Demonstration 22
Requirements 22
Setup 22
Settings 23
Procedure 23
Summary 24
Appendix A—Reference 26
Appendix B—Flowcharts 27
Appendix C—Source Code 29
C File 29
Header File 32
HTML File 33
Java Source Code File 34

List of Figures

Figure 1. General Setup with the eZ80F91 MCU as a Webserver 2
Figure 2. Communication between Client Browser and the eZ80F91 Webserver 4
Figure 3. LED and Switch Control Demo Webpage 5
Figure 4. Java Applet Component on HTML Webpage 5
Figure 5. Flowchart to Toggle the LEDs on Development Board using Buttons 27
Figure 6. Flowchart to Change the Color of LED Icons on Webpage using Switches 28

List of Tables

Table 1. List of References 26

Abstract

An Internet-enabled product built around eZ80Acclaim!™ family of microcontrollers using the ZiLOG TCP/IP stack (ZTP) involves developing webpages to control its functionality. This Application Note focuses on developing an application to remotely control the pins of the eZ80F91 MCU from a webpage, and display the status of a pin on the webpage. The details on how to write a Java applet, add the applet into a HTML code, use ZTP's HTTP-related APIs and integrate the complete system are discussed in this Application Note.

With this Java-based application, a two-way control mechanism is demonstrated wherein the user controls the LEDs on the eZ80F91 Development Board via the buttons on the webpage, and toggles the color of indicators (LED icons) on the webpage using switches on the Development Board.

This Java-based Webpage Interface is thus another Java-enabled process control application that uses the Internet/Intranet. (See related ZiLOG Application Note on a Thermostat Demo using the eZ80F91 MCU.)

The source code file (saved in the WinZip format) associated with this Application Note is `AN0178-SC01.zip`, and is available on the ZiLOG website.

ZTP Overview

Figure 1 depicts the general data communication path using the eZ80F91 Development Kit, the ZPAKII, and an Ethernet Hub. With this hardware, the eZ80F91 MCU is ready to be used as an efficient webserver when ZiLOG's ZTP software is downloaded on it. ZTP provides the software to drive the hardware used for TCP/IP connections. This hardware comprises SERIAL1 (UART1/UART2) for PPP connections and the Ethernet Media Access Controller (EMAC) for Ethernet connections among other peripherals.

The ZiLOG TCP/IP Software Suite (ZTP) contains a set of libraries that implement an embedded TCP/IP stack and includes a pre-emptive, multi-tasking kernel.

The ZTP Applications Programming Interface (API) allows programmers using any member of the eZ80® family of processors (including the eZ80Acclaim! product line) to rapidly develop internet-ready applications with minimal effort.

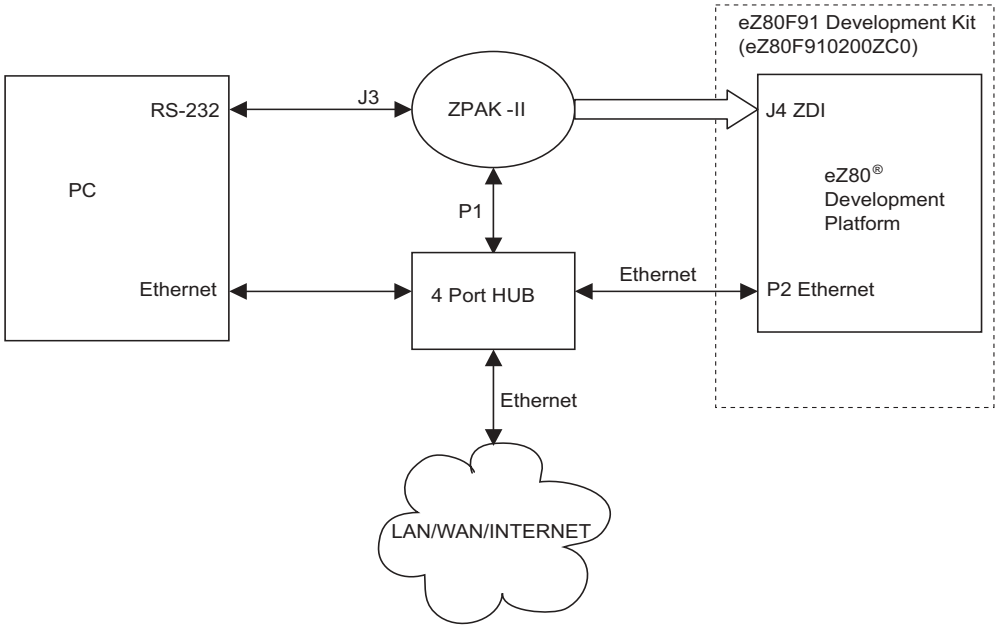

Figure 1. General Setup with the eZ80F91 MCU as a Webserver

HTTP Server in ZTP

The implementation of Hyper TextTransfer Protocol (HTTP) in ZTP enables eZ80F91 microcontroller to work as an HTTP server. The eZ80F91 MCU-based products implemented as an HTTP server can store webpages and provide them to browsers on demand. The HTTP layer also provides the facility of creating socket connections between networked products and allows them to interchange data reliably over the Ethernet.

For this Java-based Webpage Interface application, a C program supports the webpages and helps the user interact with the webserver.

Using the ZTP HTTP user interface primarily involves writing user application code that calls the appropriate ZTP HTTP API functions. It also involves integrating user webpages into the webserver using ZDSII.

ZTP HTTP CGI Functions

Embedded systems typically do not contain a file system. The lack of a file system means that embedded systems cannot save CGI scripts as separate `*.cgi` files.

Instead of saving CGI scripts as separate `*.cgi` files, ZTP uses C function calls, collectively called CGI functions. When a CGI function is called, the HTML code stored in the webserver is sent to the web browser. It is in these function calls that

a programmer writes code to read the information sent by a web browser. This information is then processed as required by the application.

ZTP provides the following CGI functions to the user.

- `int http_output_reply(http_request *request, int reply)`
- `char *http_find_argument(http_request *request, char *arg)`
- `int _http_write(http_request *request, char *buff, int count)`

In each ZTP CGI function, the pointer to the `request` structure is used to keep the requests from different clients separate.

The function `http_output_reply()` is used to return an acknowledgement to the browser that made the request.

The `http_find_argument`() function is used to extract parameters from the received data in the parsed browser request.

The macro `_http_write`() is used to return data to the browser that sent the request that invoked the CGI function.

For detailed information on the protocols used in ZTP, refer to the ZTP Programmer's Guide (RM0008), available along with the ZTP software.

Java Applets and HTML

A Java applet is a program that adheres to a set of conventions that allow it to run on Java-compatible browsers. Most of the standard HTML browsers support Java applet execution. The standalone systems using Java applets must add a Java Virtual Machine (JVM) to run an applet.

Hypertext Markup Language (HTML) is used to display information in webbrowsers. A webpage is basically coded in HTML, which is interpreted at run-time by the browser. A webpage that can change its content depending on the information obtained by the server at run-time, is called dynamic. A static page on the other hand displays fixed information. Inserting an applet into the webpage allows the webpage to display dynamic information because the applet changes its content during run-time. The web browser creates the applet object when it encounters the `<applet>` tag in the HTML code; the `<applet>` tag provides the name of the Java class file (stored as Java byte code in the webserver) from which the applet object is created. Visit www.java.sun.com for information on the Java language.

Developing the Java-based Webpage Interface Application

This section discusses the implementation of the Java-based Webpage Interface in detail. The implementation of the operations that occur when a button is clicked on the webpage and when a switch on the Development Board is pressed are

explained in the following sections. The section also includes instructions on how to interface the application specific files to the standard ZTP files.

The block diagram of eZ80F91 webserver setup for the Java-based Webpage Interface application is shown in Figure 1. The eZ80F91 MCU is configured to work as a webserver, and a client PC with a browser is used to control the GPIO pins on the eZ80F91 MCU via the webpages. The switches SW1, SW2 and the LEDs DCD (D1), RX (D2) are on the Development Board and are used to demonstrate the working of this application.

A browser is used to access the eZ80F91 webserver to obtain the current status of different GPIO pins or to change the status of the pins and thereby control the system over the Ethernet. The locally networked devices are connected together in an intranet and are accessible to the outside world (Internet) through a gateway.

An overview of the communication between eZ80F91 webserver and an HTML browser is illustrated in Figure 2.

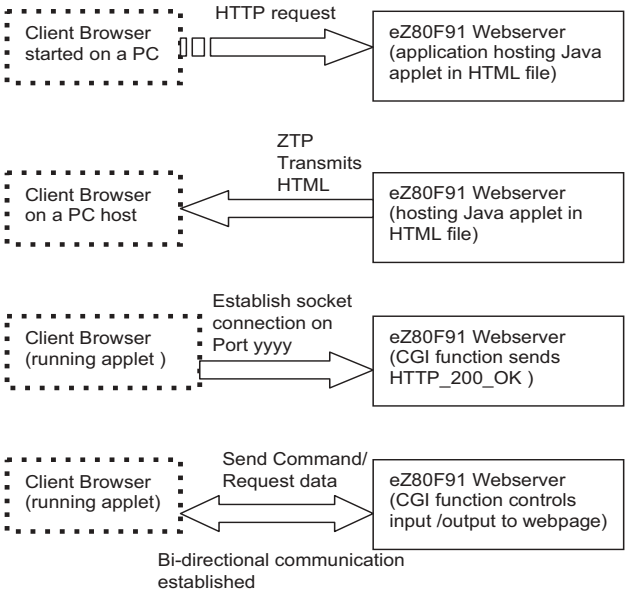
Client Browser started on a PC
HTTP request
eZ80F91 Webserver (application hosting Java applet in HTML file)
ZTP Transmits HTML
Client Browser on a PC host
eZ80F91 Webserver (hosting Java applet in HTML file)
Establish socket connection on Port yyyy
Client Browser (running applet)
eZ80F91 Webserver (CGI function sends HTTP_200_OK)
Send Command/ Request data
Client Browser (running applet)
eZ80F91 Webserver (CGI function controls input /output to webpage)
Bi-directional communication established

Figure 2. Communication between Client Browser and the eZ80F91 Webserver

The webserver transmits an HTML file (containing a link to the Java applet) to the client browser. The Java applet contains a graphical user interface (GUI) component that is displayed on the webpage as shown in Figures 3 and 4. The webpage displays the buttons that control the LEDs and the indicators (LED icons) that are controlled by the switches on the Development Board.

Figure 3. LED and Switch Control Demo Webpage

Figure 4 is a close-up of the Java applet display.

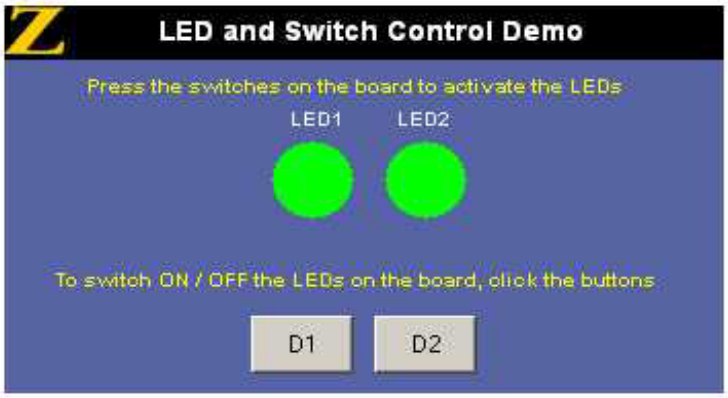
LED and Switch Control Demo
Press the switches on the board to activate the LEDs
LED1
LED2
To switch ON / OFF the LEDs on the board, click the buttons
D1
D2

Figure 4. Java Applet Component on HTML Webpage

Note: By default, the HTTP server closes the TCP connection as soon as the HTTP request is processed and therefore persistent connections are not possible.

The software implementation can be divided into the following sections:

- Designing the webpage, which includes coding the Java applet for the GUI.
- Developing the functionality, which comprises the following:
  - Implementing the functionality to control the LEDs on the Development Board
  - Implementing the functionality to effect a change in the indicators (LED icons) on the webpage using the switches on the Development Board

Creating the Webpage

The source code for the HTML file `Switch.htm` displays the webpage as shown in Figure 3. The HTML script contains a link to the Java applet using the lines of code as presented below:

```
<applet width="340" height="250" code="SwitchCtrl.class">
<param name="bgcolor" value="#FFFFFF">
</applet>
```

The applet is executed from the Java class file `SwitchCtrl.class` which is previously stored in the eZ80F91 webserver in the form of ASCII characters. The `SwitchCtrl.class` file is obtained upon compiling the `SwitchCtrl.java` file. The Java file `SwitchCtrl.java` is coded as detailed below. The entire Java source code is available in Appendix C on page 29.

Importing Relevant Java Packages

In the `SwitchCtrl.java` file, the relevant packages that the main program uses are imported in the beginning of the file. The packages are similar to the `*.lib` and `*.h` files of the C language. In the program for the applet, the following packages containing the relevant classes are defined as imported:

```
import java.applet.Applet;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.awt.event.*;
```

The imported packages are part of the core Java APIs that every Java program uses in the Java run-time environment. The `java.applet` package contains classes that are essential for Java applets. The `java.awt` package contains the most frequently used classes in the Abstract Window Toolkit (AWT), which provides the Java graphical user interface (GUI) properties.

A sub-class to extend the properties provided to the applet by the packages is implemented next, with the following lines of code:

```
public class SwitchCtrl extends Applet implements Runnable ,
ActionListener{
...
)
```

The `extends` keyword indicates that `SwitchCtrl` is a subclass of the `Applet` class.

From the Applet class, applets inherit a great deal of functionality, the most important of which is the ability to respond to browser requests. When a Java-capable browser loads a page containing an applet, the browser sends a request to the applet telling the applet to initialize itself and start executing.

Declaring the Variables

After creating the Applet class, the variables that can be used by any function or sub class are declared within the `SwitchCtrl` class as follows:

```
...
Socket rwSocket;
Socket inSocket;
Thread readThread;
String ipaddr=null;
String ip=null;
int portNumber;
int port;
Button xButton;
Button yButton;
DataOutputStream os;
DataInputStream is;
```

```
Image image1;
Color firstBulbColor = Color.green;
Color secondBulbColor = Color.green;
...
```

Initializing the Applet

The applet is initialized using the `public void init()` method. The `init()` method is called when the applet is first created and loaded by the underlying software (the web browser). This method performs all the one-time operations that the applet requires for its operation such as creating the user interface or setting the font or displaying the logo.

```
...
public void init(){
...
}
```

The file formats `*.gif` and `*.jpg` are used to load and display images on the browser screen. Images are stored as an instance of the `Image` class. The applet method `getImage()` returns an instance of these files.

```
image1 = getImage(getDocumentBase(), "logo.gif");
```

Setting the Applet Background Colour

The following piece of code is used to set the background color of the applet based on the color value received from the HTML page.

```
...
String s3 = getParameter("bgcolor");
if(s3 != null)
try{
   setBackground(new Color(Integer.parseInt(s3.substring(1),
   16)));
}
...
```

The `catch()` method that is used to handle errors and exceptions in Java is used to handle invalid color values. The error message is displayed on the Java Console that can be opened from the web browser.

Note: The Java Console is available as an option under **Tools** in the web browser when the Java Runtime Environment (JRE) is installed on the client PC. The JRE can be downloaded from the website, http://www.java.sun.com.

```
...
catch(NumberFormatException _ex){
   System.out.println("Invalid format for bgcolor: " + s3);
}
setLayout(null);
...
```

Creating the Buttons

To create buttons with text, the method `new Button()` is used. The `setBounds()` method specifies the area (the X and Y axis) where the button is to be placed on the webpage and also specifies its width and height.

The `add()` method adds the new button to the Applet. The `setEnabled()` method either activates or disables the button. The button is activated only when the `setEnabled()` method is `true` and disabled when it is `false`. The following code illustrates the methods explained to create buttons.

```
...
xButton = new Button("D1");
xButton.setBounds(120, 160, 50, 30);
xButton.addActionListener(this);
add(xButton);
yButton = new Button("D2");
yButton.setBounds(180, 160, 50, 30);
yButton.addActionListener(this);
add(yButton);
xButton.setEnabled(true);
yButton.setEnabled(true);
```

```
...
```

Obtaining the Host IP Address

The applet is started using the `public void start()` method. The IP address of the host where the applet is run must be obtained so that a socket can be opened for a communication channel to be set up between the client and the webserver. The following code is used to set the port number and obtain the host IP address, which is then sent to the webserver (see the section on Adding and Integrating Application Specific Files to ZTP on page 16).

```
...
public void start(){
        System.out.println("Enter Start");
        ipaddr = getCodeBase().getHost();
        System.out.println("IP Addr   " + ipaddr.toString());
        portNumber = 80;
        System.out.println("PortNumber=" + portNumber);
```

The following code handles the error due to a connection failure.

```
try{
   rwSocket = new Socket(ipaddr.toString(), portNumber);
}catch(UnknownHostException _ex){
   System.out.println("An error occurred while establishing the
   communication with the server");
}catch(IOException _ex){
   System.out.println("Couldn't get I/O for the connection to: "
   +ipaddr);
}catch(Exception e){
   System.out.println("An error occurred while reading the
   response.Try reloading the page. ");
}
```

A new thread is then created and started with the following lines of code:

```
if(readThread == null){
```

```
    readThread = new Thread(this);
    readThread.start();
}
```

Creating the Applet Heading

The text or the heading for the Applet is printed using the `paint(Graphics)` method of the `java.awt.Graphics` package. The `paint(Graphics)` method is coded as follows:

```
public void paint(Graphics g) {
    System.out.println("Enter Graphics"+g);
    g.setColor(Color.black);
    g.fillRect(0,0,400,28);
    g.drawImage(image1,0,0, this);
    g.setColor(Color.white);
    g.setFont(new Font("Arial", 1, 14));
    g.drawString("LED and Switch Control Demo", 75, 18);
    g.setColor(Color.yellow);
    g.setFont(new Font("Arial", 1, 10));
    g.drawString("Press the switches on the board to activate the
    LEDs below", 40, 45);
    g.setColor(Color.white);
    g.setFont(new Font("Arial", 1, 10));
    g.drawString("LED1", 139, 62);
    g.drawString("LED2", 191, 62);
    g.setColor(firstBulbColor);
    g.fillOval(130,70,40,40);
    g.setColor(secondBulbColor);
    g.fillOval(185,70,40,40);
    g.setColor(Color.yellow);
    g.setFont(new Font("Arial", 1, 10));
    g.drawString("To switch ON / OFF the LEDs on the board, click
    the buttons", 25, 145);
    System.out.println("Exit Graphics");
}
```

The `setFont()` method is used to set the font. The `drawString()` method prints the specified text on the browser. The `fillOval()` method paints the LED icons (labelled as LED1 and LED2 on the webpage) on the screen according to the position (X and Y axis) and size (width and height) as specified. The `setColor()` method is used to specify the color for the LED icon.

Synchronizing Independent Threads

To run the main thread and obtain the IP address of the client at Port 5000 (user-specified), the following code is used. The `synchronized` keyword tells the applet to synchronize all the threads.

```
synchronized public void run() {
   System.out.println("Enter RUN");
   ip = getCodeBase().getHost();
   System.out.println("IP Addr   " + ip.toString());
   port = 5000;
```

Reading and Parsing the Data from the Port

When the applet runs, the browser displays the indicators (LED icons, LED1 and LED2) and buttons (D1 and D2) on the screen.

The `Socket` class is used to set up Port 5000 for input. The data on the port is then read and parsed to obtain the proper string value after eliminating `&` and `\n` characters. This value is then passed to the method `updateBulbStatus()`.

```
...
try{
   inSocket = new Socket(ip.toString(),port);
   is = new DataInputStream(inSocket.getInputStream());
   System.out.println("InStream is:>> " +is);
   while(true) {
      str = "";
   while((b = is.readByte()) != '&') { }
   while ((b = is.readByte()) != '\n') {
      str = str + (char)b; //read bytes, not Unicode characters
   }
   System.out.println("Input read is:>>" +str);
```

```
   updateBulbStatus(str);
   }
}catch(Exception e) {
   System.out.println("An error occurred while reading the
   response.Try reloading the page. ");
}
...
```

Changing the Color of the Indicators

The `updateBulbStatus()` method, which toggles the color of the indicators (LED icons) on the webpage between red and green, is constructed as illustrated below:

```
public void updateBulbStatus(String msg){
   System.out.println("Enter UpdateBulbStatus");
   System.out.println("Received String" +msg);
   if (msg.equals("LED1on")) {
      redraw(1,Color.red);
   }
   else if (msg.equals("LED1off")){
      redraw(1,Color.green);
   }
   else if (msg.equals("LED2on")){
      redraw(2,Color.red);
   }
   else if (msg.equals("LED2off")){
      redraw(2,Color.green);
   }
   System.out.println("Exit updateBulbStatus");
}
...
```

Controlling the LEDs on the Development Board

When the button D1 or D2 is clicked on the webpage, the `actionPerformed()` method is invoked and a data string corresponding to the clicked button is stored in variable `s`.

```
...
public void actionPerformed(ActionEvent ev) {
   System.out.println("Enter Action Performed");
   String label = ev.getActionCommand();
   String s = new String();
   if(label.equals("D1")){
      s = "command=1&dummy=0";
      sendData(s);
   }else if(label.equals("D2")){
      s = "command=2&dummy=0";
      sendData(s);
   }
   System.out.println("Exit Action Performed");
}
```

The data in the string `s` is sent to the socket by the `sendData()` method with an alert for any error that may occur. After sending the data, the socket is closed to complete the communication. The HTTP layer of the webserver listening on Port 5000 parses and interprets this data and provides the requisite value to the CGI function `switchinput_cgi`.

```
...
public void sendData(String s) {
   System.out.println("Enter SendData");
   String s1 = null;
   s1 =s;
   s1 = "GET /switch_cgi?" + s1 + " HTTP/1.1\r\n\r\n";
   System.out.println("Data to send =" + s1);
   try{
      System.out.println("rwSocket::"+rwSocket);
```

```
        Socket outSocket = new Socket(ipaddr, portNumber);
        os = new DataOutputStream(outSocket.getOutputStream());
        os.flush();
        os.write(s1.getBytes());
        outSocket.close();
    }catch(IOException _ex){
System.err.println("Write exception"+_ex);
}
System.out.println("Exit Send Data");
}
```

The Java source code construction for the applet is now complete. To compile this code, the Java compiler version 1.1 must be installed on the PC (PCs with a web browser such as Internet Explorer or Netscape already contain the Java VIrtual Machine that contains the Java compiler). The following command, at the DOS prompt, is used to compile the Java code:

```
Javac –deprecation –target 1.1 SwitchCtrl.java
```

After successful compilation, a class file `SwitchCtrl.class` is created. This class file is further converted to a `SwitchCtrl_class.c` file (where the contents of the file are stored in the form of an array) when the project is compiled using the ZiLOG's ZDSII compiler.

Controlling the LEDs on the Development Board

In the Java-based Webpage Interface application a CGI function, `switchinput_cgi()` is used to call the ZTP HTTP CGI functions in order to read the user input from the HTML webpage when the button D1 or D2 is clicked on the webpage, and to transmit that input data to the Development Board. See the section ZTP HTTP CGI Functions on page 2.

The `switchinput_cgi()` function is declared as follows:

```
int switchinput_cgi(struct http_request *request)
```

This CGI function initially calls the ZTP HTTP API `http_init()` function to initialize the webserver, makes a TCP connection on the specified port and waits for a client request. Upon receiving a request from the client, the webserver provides a response according to the webserver configuration.

The CGI function then calls the ZTP HTTP CGI function, `http_output_reply()` to send an acknowledgement to the client browser. The `http_find_argument()` is called to search for the string command indicating which of the buttons (D1 or D2) is pressed; the respective LED on the Development Board is then switched on/off. See the flowchart illustrating the sequence of events in Figure 5 on page 27.

Controlling the LED Icons on the Webpage

The Port B of the eZ80F91 MCU is initialized with PB0 and PB1 as input pins and PB3 and PB4 as output pins (see `Switch.c` file in the `AN0178-SC01.zip` file available on the ZiLOG website). Port B is continuously polled for any change in status in ZTP's `main.c` file.

When the SW1/SW2 switch on the Development Board is pressed, the change in the switch status is captured by the `Switch_Input()` function (a flag is set or reset to indicate the status of the switches) and an appropriate string is written for the switch. The strings `LEDon` or `LEDoff` are used to indicate whether the LED icons on the webpage must be displayed in green (OFF) or red (ON). These strings are sent as arguments in the `switchinput_cgi()` function. See Controlling the LEDs on the Development Board section on page 15 for details of this CGI function.

The ZTP HTTP API `http_find_argument()` is called within the `switchinput_cgi()` function to search for the string (`LED1on`/`LED1off`). This string is sent to the browser where the color of the indicators (LED icons, LED1/LED2) on the webpage turns green or red. See the flowchart illustrating the sequence of events in Figure 6 on page 28.

Adding and Integrating Application Specific Files to ZTP

The Java-based Webpage Interface application described in this Application Note requires the eZ80® Development Board with the eZ80F91 MCU, and the ZiLOG TCP/IP stack (ZTP). To execute this application, the files specific to the application must be added and integrated to the ZTP stack before the stack is downloaded onto the eZ80F91 MCU. This section contains the details of adding the application specific files to the ZTP stack.

The Java-based Webpage Interface files are in the `AN0178-SC01.zip` file available on the ZiLOG website. The application files are of the following types:

- C (*.c) files
- Header (*.h) files
- HTML (*.htm/*.html) files
- Java (*.class) files

- Image (*.gif) files

The ZTP stack is available on the ZiLOG website and can be downloaded to a PC with a user registration key. ZTP can be installed in any location as specified by the user; its default location is `C:\Program Files\ZiLOG`.

Note: Before adding and integrating the application specific files to ZTP, ensure that all the settings for the ZTP stack are at default values.

Perform the following steps to add and integrate the Demo files to the ZTP stack:

1. Download ZTP, browse to the location where ZTP is downloaded, and open the `\website.Acclaim` folder.
2. Download the `AN0178-SC01.zip` file and extract its contents to a folder on your PC (this folder is referred to as `\JavaWebControl` folder in the rest of the Application Note). Notice the two extracted folders within the `\JavaWebControl` folder:

   `\JWC_Demo`

   `\JWC_Website.Acclaim`

3. Select and copy the `*.htm, *.class, *.h` and the `*.gif` files in the `\JavaWebControl\JWC_Website.Acclaim` folder and paste them into the `..\ZTP\website.Acclaim` folder.
4. Select and copy all the `*.c,` and `*.h,` files located in the `\JavaWebControl\JWC_Demo` folder and paste them into the `..\ZTP\Demo` directory.
5. Launch ZDSII and open the project file, `website.pro` located in the path: `..\ZTP\website.Acclaim`.
6. Now add the `*.htm, *.class, *.h` and the `*.gif` files located in the `..\website.Acclaim` folder to the project, using the sequence of steps: **Project** → **Add Files**.

   The `*.htm, *.class, *.h` and the `*.gif` files to be added are listed below:

   `Switch.htm`

   `SwitchCtrl.class`

   `switch.h`

   `logo.gif`

7. Open the website.c file from within ZDSII, and enter the following into it:

```
#include <Switch.h>

//HTML
extern struct staticpage switch_htm;

//CGI
int switchinput_cgi(struct http_request *request);

//Java applets
extern struct staticpage SwitchCtrl_class;

//for the image
extern const struct staticpage logo gif;

//for LED status
static unsigned char ledDCDstatus = 1;
static unsigned char ledRXstatus = 1;
```

8. The website.c file contains the array, Webpage website[], with information on the HTML pages. Replace the last line of the array, {0, NULL, NULL, NULL }, with the following lines:

```
{HTTP_PAGE_STATIC, "/switch.htm","text/html", &switch_htm},
{HTTP_PAGE_DYNAMIC, "/switch_cgi", "text/html",(struct
staticpage *)switchinput_cgi},
{HTTP_PAGE_STATIC, "/logo.gif", "image/gif", &logo_gif},
{HTTP_PAGE_STATIC, "/SwitchCtrl.class", "application/octect-
stream", &SwitchCtrl_class},
{0, NULL, NULL, NULL};
```

9. Add the following definition for the switchinput_cgi() function at the end of the website.c file.

```
// Defintion for CGI Function
int switchinput_cgi(struct http_request *request)
{
   char *switchstr;
   int switchval;
   http_output_reply(request,HTTP_200_OK);

   // Get a pointer to string from Web page
   switchstr = http_find_argument(request, (unsigned char*)
                                    "command");

   switchval = atoi(switchstr);// Get the Switch number
   if (switchval == 1)       // If SW1 icon clicked
   {
      if (!ledDCDstatus)     // Check status of LED
      {
         PB_DR |= 0x08;      // PB3 set to High, DCD LED glows
         ledDCDstatus = 1;   // Set Flag
      }
      else
      {
         PB_DR &= 0xF7;      // PB3 reset Low, DCD LED put off
         ledDCDstatus = 0;   // Reset Flag
      }
   }
else if (switchval == 2)
{
   if (!ledRXstatus)
   {
      PB_DR |= 0x10;         // PB4 set to High, RX LED glows
      ledRXstatus = 1;       // Set Flag
   }
   else
   {
```

```
        PB_DR &= 0xEF;          // PB4 reset Low, RX LED put off
        ledRXstatus = 0;        // Reset Flag
    }
}
return( OK );
}
```

10. From within ZDSII, open the `left.htm` file located in the `\Web Files` folder. Search for **CGI Calculator** and locate the following line:

```
   <a href="switch.htm" target="_top">CGI
Calculator</a><br>
```

11. Add the following piece of HTML code below the `CGI Calculator` line, to create a link from the default eZ80Acclaim! webpage to the Java-based Webpage Interface Demo webpage.

```
   <a href="switch.htm" target="main"> Java Web
Control </a><br>
```

12. Build the `website.pro` project to obtain the new library file, `Acclaim_website.lib`.
13. Copy the `Acclaim_website.lib` library file to the path: `..\ZTP\libs.`

Note: Please note that the `..ZTP\libs` folder already contains an `Acclaim_website.lib` file and it must be replaced with the newly generated file. Click **Yes** to replace the file.

14. Close the `website.pro` project.
15. In ZDSII, open the `AcclaimDemo.pro` file available in the path: `..\ZTP\Demo.`
16. Add the `*.c` file located in the `\JavaWebControl\JWC_Demo` folder to the project, using the sequence of steps: **Project** → **Add Files**.

    The `*.c` file to be added is `Switch.c.`

17. Open the `main.c` file of the AcclaimDemo project and add the following include file:

```
#include <switch.h>
```

18. In the `main.c` file, observe the following `BootInfo` structure definition:

```
struct BootInfo Bootrecord = {
        "192.168.1.1",/* Default IP address */
        "192.168.1.4",/* Default Gateway */
        "192.168.1.5",/* Default Timer Server */
        "192.168.1.6",/* Default File Server */
        "",
        "192.168.1.7",/* Default Name Server */
        "",
        0xffffff00UL/* Default Subnet Mask */
        };
```

The `Bootrecord` variable contains the network parameters and settings (in the four-octet dotted decimal format) that are specific to the local area network at ZiLOG, as default.

Modify the above structure definition with appropriate IP addresses within your local area network.

19. Add the following lines of code at the end of the `main.c` file, before the line `return OK;` to call the `Switch_Input()` function at intervals of one second:

```
PortBInit();                    //Initialize pin PB0 as Input

//Create a start task
resume(create((procptr)task1,1024,20,”task1”,0,0));
while(1)                        //Loop forever
{
   Switch_Input();              //Check status of SW1 and SW2
   sleep(1);
}
```

20. Open the `eZ80_HW_Config.c` file and change the default MAC address (provided by ZTP) such that each eZ80 Development Board on the LAN contains a unique MAC address. For example:

```
const BYTE f91_mac_addr [EP_ALEN] = {0x00, 0x90, 0x23, 0x00,
0xDF, 0x91};
```

In the 6 byte MAC address shown above, the first three bytes must not be modified; the last three bytes can be used to assign a unique MAC address to the eZ80 Development Board.

21. Open the `ipw_ez80.c` file. For this application, DHCP (Dynamic Host Configuration Protocol) is disabled; therefore, ensure the following:

```
b_use_dhcp = FALSE
```

22. Save the files and close the AcclaimDemo project.

Demonstration

This section contains the requirements and instructions to set up the Java-based Webpage Interface Demo and run it.

Requirements

The requirements are classified as hardware and software.

Hardware

- eZ80F91 Development Kit (eZ80F910200ZCO)
- PC with an Internet browser with the Java Virtual Machine

Software

- ZiLOG Developer Studio II—IDE for eZ80Acclaim! (ZDSII)
- ZiLOG's TCP/IP stack (ZTP)

Setup

The basic setup to assemble the Java-based Webpage Interface demo is illustrated in Figure 1 on page 2. This setup illustrates the connections between the PC, ZPAKII, LAN/WAN/Internet and the eZ80F91 Development Kit.

Settings

HyperTerminal Settings

- Set HyperTerminal to 57.6 Kbps Baud and 8-N-1, with no flow control

Jumper Settings

For the eZ80F91 Development Board

- J11, J7, J2 are ON
- J3, J20, J21, J22 are OFF
- For J14, connect 2 and 3
- For J19, MEM_CEN1 is ON, and CS_EX_IN, MEM_CEN2, and MEM_CEN3 are OFF

For the eZ80F91 Module on eZ80F91 Development Board

- JP3 is ON

Additional Settings for the eZ80F91 Development Board

- Connect PB3 of J6 Header to Pin6 of the J9 Connector. Count Pin6 starting from the J9 marking.
- Connect PB4 of J6 Header to Pin7 of the J9 connector. Count Pin7 starting from the J9 marking.

Procedure

The procedure to build and run the Java-based Webpage Interface Demo is described in this section.

1. Ensure that the required Java-based Webpage Interface files are added and integrated to ZTP before proceeding. See section Adding and Integrating Application Specific Files to ZTP on page 16, for details.
2. Make the connections as per Figure 1. Follow the jumper settings provided in the section on Jumper Settings above.
3. Connect the 9-volt power supply to the eZ80F91 Development Kit.
4. Connect the 5-volt power supply to ZPAKII and the 7.5-volt power supply to the Ethernet Hub.
5. Launch the HyperTerminal and follow the settings provided in the HyperTerminal Settings section above.

6. From within the HyperTerminal, press *z* repeatedly, and then press the reset button on ZPAKII to view the menu to set the ZPAKII IP address.
7. Enter H to display help menu, and follow the menu instructions to obtain the IP address for ZPAKII in order to download the Demo file. This ZPAKII IP address must be entered in the ZDSII.
8. Launch ZDSII–for eZ80Acclaim! and open the Java-based Webpage Interface project file (`AcclaimDemo.pro`) located in the path: `..\ZTP\Demo.`
9. Open the `main.c` file. Ensure that the `BootInfo` structure contains information that is relevant to your network configuration. Use the IP address in the structure to browse the Internet to view the Web Switch Demo web pages.
10. Build the project and download the resulting file to the eZ80F91 Module on the eZ80 Development Board, using ZDSII.
11. Run the Java-based Webpage Interface Demo. Refer to Running the Java-based Webpage Interface Demo section below.

Running the Java-based Webpage Interface Demo

1. Launch the Internet browser on the PC. Enter the IP address for the eZ80F91 Development Board specified in `main.c`. The `Index.html` page is displayed. The LEDs (labeled DCD and RX) on the Development Board light up to indicate an active link between the eZ80F91 webserver and the PC-based client browser.
2. Click on the link **Java Web Control** located in the left side of the screen. A new web page (**LED and Switch Control Demo**) opens displaying the buttons and the indicators (LED icons).
3. Click on the button **D1.** Notice that the LED (labeled DCD) on the Development Board is switched off and this on/off feature toggles on every mouse click. The same procedure is performed on the **D2** button for the LED labeled Rx.
4. Press the switch **SW1/SW2** on the eZ80F91 Development Board. The color of the indicator (LED icons) changes from red to green. Subsequent clicking of the switches toggles the color of the LED icons on the webpage.

Summary

The eZ80F91 microcontroller is used for building Internet-enabled products that are controlled over an Internet or Intranet using webpages. This Application Note demonstrates how the current status of a pin on the eZ80F91 MCU can be

remotely viewed and controlled by using a Java applet in a webpage. This type of simple interface utilizes the HTTP component of ZiLOG's ZTP stack and provides reliable transfer and control of data. The details provided in this Application Note can be used to build more complex user interfaces for real world applications.

Appendix A—Reference

Further details about the eZ80F91 MCU, eZ80 CPU, and ZTP can be found in the references listed in Table 1.

Table 1. List of References

Topic	Document Name
eZ80F91 MCU	eZ80F91 Development Kit User Manual (UM0142)
	eZ80F91 Flash MCU with Ethernet MAC Product Specifications (PS0192)
	eZ80F91 Module Product Specifications (PS0193)
eZ80 CPU	eZ80 CPU User Manual (UM0077)
ZiLOG's TCP/IP Software Suite (ZTP)	ZiLOG's TCP/IP Software Suite Programmer's Guide Reference Manual (RM0008)

Appendix B—Flowcharts

This appendix contains the flowcharts for the Java-based Webpage Control Interface implementation using the eZ80F91 MCU.

Figure 5 illustrates the flowchart for toggling the LEDs on the eZ80F91 Development Board using the D1 and D2 buttons on the webpage.

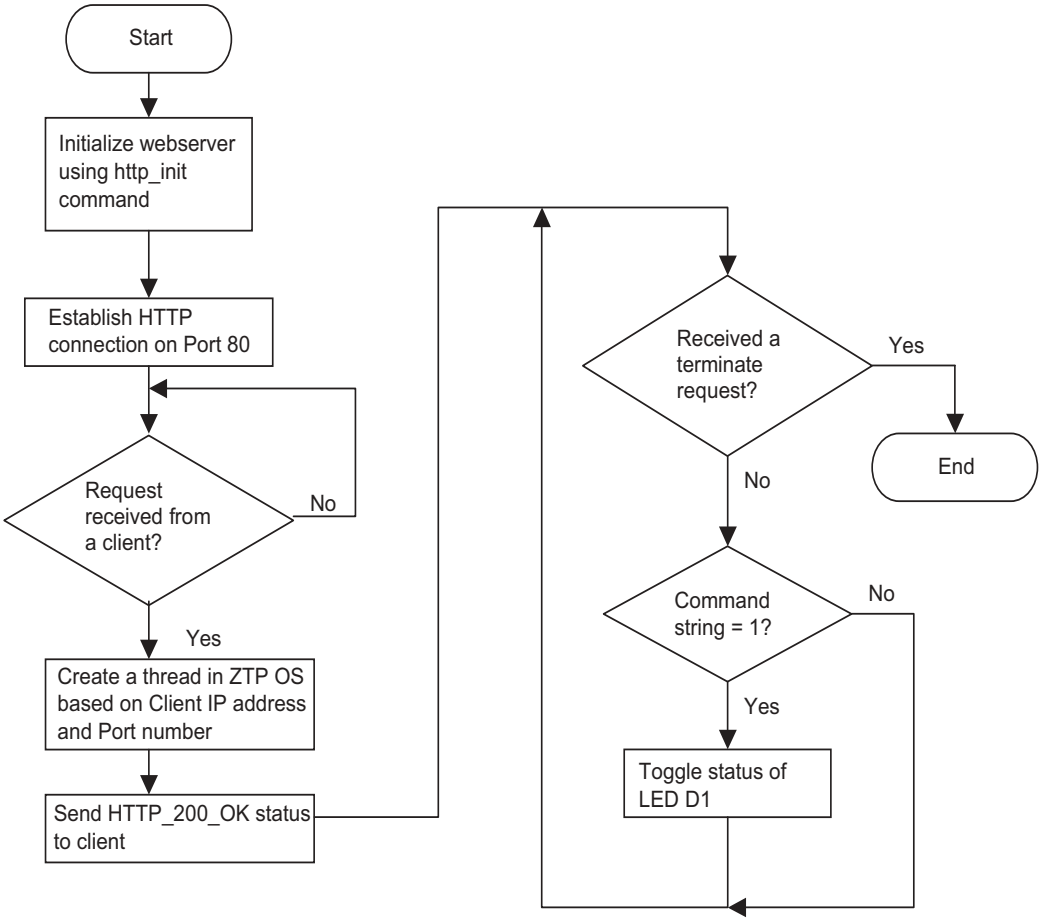

Figure 5. Flowchart to Toggle the LEDs on Development Board using Buttons

Figure 6 illustrates the flowchart for changing the color of the LED icons (LED1 and LED2) on the webpage using the SW1 and SW2 switches on the eZ80F91 Development Board.

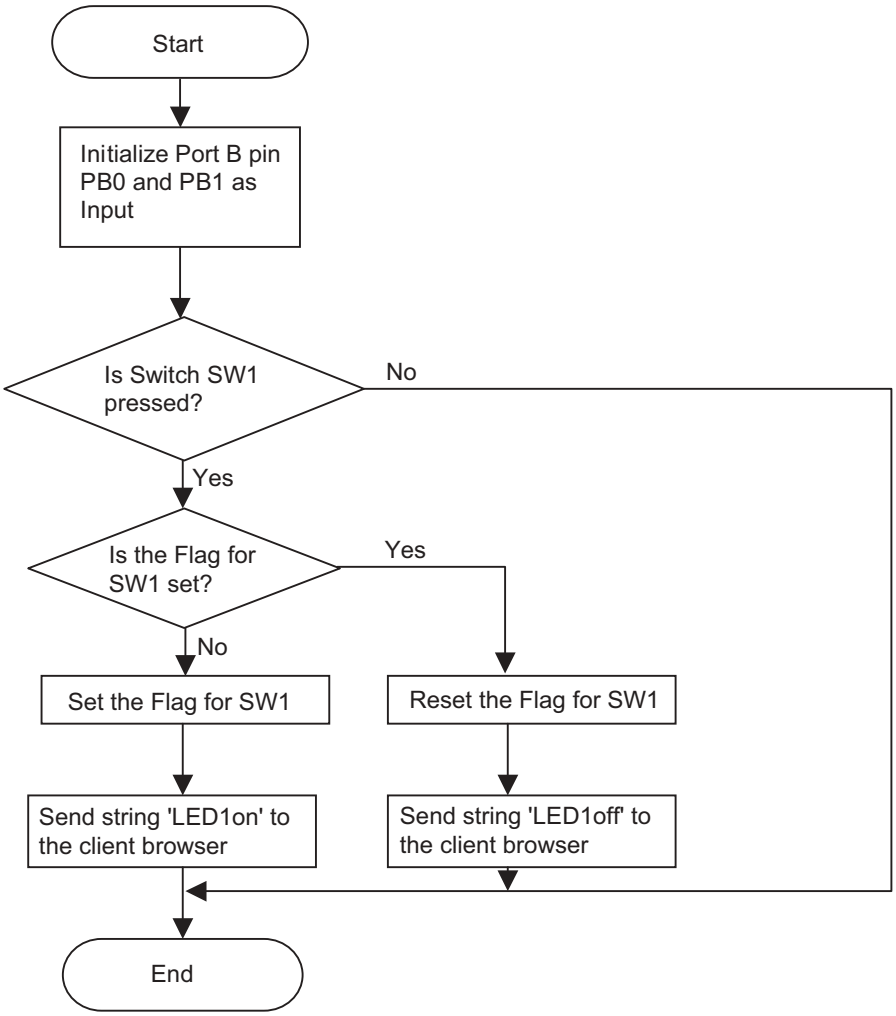

Figure 6. Flowchart to Change the Color of LED Icons on Webpage using Switches

Appendix C—Source Code

This appendix provides a listing of the source code associated with this Application Note. The source code file AN0178-SC01.zip is available on the ZiLOG website.

C File

Following C file is listed in this section:

- Switch.c

```
/*
************************************************************
* File:   Switch.c
* Description: Function definition for Switch input.
*
* Copyright 2004 ZiLOG Inc.  ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
************************************************************
*/

#include <kernel.h>
#include <stdlib.h>
#include <tcb.h>
#include <dgram.h>
#include <switch.h>                         // Include header file


volatile char read_switchSW1;               // Switch SW1 status
volatile char read_switchSW2;               // Switch SW2 status
unsigned char switchon1;                    // Flag indicator for Switch SW1
unsigned char switchon2;                    // Flag indicator for Switch SW2

void PortBInit(void)                        // Initialize pin PB0
{
   PB_DR = 0x1B;              // PB0,PB1 = Input, PB3,PB4 = Output
   PB_DDR = 0x03;
```

```
   PB_ALT1 = 0x00;
   PB_ALT2 = 0x00;
}

int task2(int cfd)
{
   while (1)                                    // Lookout for a new client
   {
      if (switchon1)
      {
         if (write(cfd,"&LED1on\n",9) == SYSERR)// Send status of SW1
                                                // to Web page
            break;
      }
      else
      {
         if (write(cfd,"&LED1off\n",10) == SYSERR)// Send default status
                                                  // (no change)
            break;
      }

      if (switchon2)
      {
         if (write(cfd,"&LED2on\n",9) == SYSERR)// Send status of SW2
                                                // to Web page
            break;
      }
      else
      {
         if (write(cfd,"&LED2off\n",10) == SYSERR)// Send default status
                                                  // (no change)
            break;
      }
      sleep(2);
   }
   close(cfd);               // Close port on client disconnect or error
}  // task2

int task1(void)
{
   int dev, cfd;
   int port = 5000;

   if((dev = open(TCP, ANYFPORT,(char *) port)) == SYSERR)// Open an
                                                // available port
      return(SYSERR);                           // Report error on problem

   control(dev,TCPC_LISTENQ,(char*)5,0);        // Open port in Listen mode
```

```
   while(1)
   {
      cfd = control(dev,TCPC_ACCEPT,0,0);       // Wait for a client
                                                // connection
      if(cfd == SYSERR)
         return SYSERR;                         // Report error on problem

      resume(create((procptr)task2,1024,20,"task2",1,cfd));
                                                // Create & start a new task
   }
}  // task1

int Switch_Input(void)
{
   int dev, cfd;
   int port = 5000;

   read_switchSW1 = PB_DR & SW1;          // Get status of SW1
   read_switchSW2 = PB_DR & SW2;          // Get status of SW2

   if(!read_switchSW1)
   {                                      // If switch SW1 is pressed
      if (!switchon1)                     // and earlier it was off
         switchon1 = 1;                   // Then put on flag
      else
         switchon1 = 0;                   // Otherwise reset flag
   }

   if(!read_switchSW2)
   {                                      // If switch SW2 is pressed
      if (!switchon2)                     // and it was off
         switchon2 = 1;                   // Then put on flag
      else
         switchon2 = 0;                   // Otherwise reset flag
   }
   return OK;
}  // Switch_Input


/*****************************************************************
*************************End of file*****************************
*****************************************************************/
```

Header File

This section contains the following header file:

- switch.h

```
/*
************************************************************
* File:   switch.h
* Description: Function declarations for Switch input.
*
* Copyright 2004 ZiLOG Inc.  ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
************************************************************
*/
#ifndef _SWITCH_H_
#define _SWITCH_H_
#include <_eZ80F91.h>

#define SW10x01              // PB0 switch1 input for LEDicon1
#define SW20x02              // PB1 switch2 input for LEDicon2

void PortBInit(void);        // Initialize pin PB0
int Switch_Input(void);      // Check SW1

int task2(int cfd);          // Prototype function
int task1(void);             // declarations

#endif

/************************************************************
*************************End of file*************************
************************************************************/
```

HTML File

The following HTML file is listed in this section:

- Switch.htm

```
<html>

<!
* File:   Switch.htm
* Description: HTML file for the Java Web Control webpage.
*
* Copyright 2004 ZiLOG Inc.  ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
-->

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title></title>
</head>

<body>
<p align="center">

<applet CODE="SwitchCtrlNew.class" WIDTH="350" HEIGHT="200">
<param name="bgcolor" value="#5563A4">
</applet>

</p>
</body>
</html>
```

Java Source Code File

This section contains the Java source code file for the applet. The following file is listed in this section:

- SwitchCtrl.java

```
// File:   SwitchCtrl.java
// Description: Source code file for the applet.
//
// Copyright 2004 ZiLOG Inc.  ALL RIGHTS RESERVED.
//
// The source code in this file was written by an
// authorized ZiLOG employee or a licensed consultant.
// The source code has been verified to the fullest
// extent possible.
//
// Permission to use this code is granted on a royalty-free
// basis. However users are cautioned to authenticate the
// code contained herein.
//
// ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.


//Import the various classes required to run the applet

import java.applet.Applet;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.awt.event.*;

public class SwitchCtrlNew extends Applet implements Runnable ,
ActionListener{

//Declare the variables

    Socket rwSocket;
    Socket inSocket;
    Thread readThread;
    String ipaddr=null;
    String ip=null;
    int portNumber;
    int port;
    Button xButton;
    Button yButton;
    DataOutputStream os;
    DataInputStream is;
```

```
    Image image1;
    String numbStr;
    Thread readThread1;
    Color firstBulbColor = Color.green;
    Color secondBulbColor = Color.green;

// Initialize the variables

public void init(){
   System.out.println("Enter init");
   image1 = getImage(getDocumentBase(), "logo.gif");
   setLayout(null);
   String s3 = getParameter("bgcolor");
   if(s3 != null)
      try {
            setBackground(new Color(Integer.parseInt(s3.substring(1),16)));
      }
      catch(NumberFormatException _ex) {
         System.out.println("Invalid format for bgcolor:" + s3);
      }
      setLayout(null);
      xButton = new Button("D1");
      xButton.setBounds(120,160,50,30);
      xButton.addActionListener(this);
      add(xButton);
      yButton = new Button("D2");
      yButton.setBounds(180,160,50,30);
      yButton.addActionListener(this);
      add(yButton);
      xButton.setEnabled(true);
      yButton.setEnabled(true);

      System.out.println("Exit init");
}

// Start the Applet

public void start(){
   System.out.println("Enter Start");
   ipaddr = getCodeBase().getHost();
   System.out.println("IP Addr" + ipaddr.toString());
   portNumber = 80;
   System.out.println("PortNumber="+portNumber);
   int SWBuffSize = 250;
   try{
         rwSocket = new Socket(ipaddr.toString(),portNumber);
   }catch(UnknownHostException _ex){
```

```
      System.out.println("An error occurred while establishing the
                          communication with the server");
   }catch(IOException _ex){
      System.out.println("Couldn't get I/O for the connection to:"+ipaddr);
   }catch(Exception e){
      System.out.println("An error occurred whilereading the response.Try
                          reloading the page.");
   }
   if(readThread == null){
      readThread = new Thread(this);
      readThread.start();
   }
   System.out.println("Exit Start");
}

//Write the text and draw the LED image

public void paint(Graphics g) {
         System.out.println("Enter Graphics"+g);
         g.setColor(Color.black);
         g.fillRect(0,0,400,28);
         g.drawImage(image1,0,0, this);
         g.setColor(Color.white);
         g.setFont(new Font("Arial",1,14));
         g.drawString("LED and Switch Control Demo",75,18);
         g.setColor(Color.yellow);
         g.setFont(new Font("Arial",1,10));
         g.drawString("Press the switches on the board to activate the LEDs",
                        40,45);
         g.setColor(Color.white);
         g.setFont(new Font("Arial",1,10));
         g.drawString("LED1",139,62);
         g.drawString("LED2",191,62);
         g.setColor(firstBulbColor);
         g.fillOval(130,70,40,40);
         g.setColor(secondBulbColor);
         g.fillOval(185,70,40,40);
         g.setColor(Color.yellow);
         g.setFont(new Font("Arial",1,10));
         g.drawString("To switch ON / OFF the LEDs on the board, click the
                        buttons",25,145);
         System.out.println("Exit Graphics");
}

public void redraw(int bulbNo,Color bulbColor) {
         if(bulbNo == 1){
            this.firstBulbColor = bulbColor;
         }else if(bulbNo == 2){
```

```
            this.secondBulbColor = bulbColor;
         }
         repaint();
}

// Stop the thread

public void stop() {
   System.out.println("Enter Stop");
   System.out.println("Exit Stop");
}

//Destroy the thread

public void destroy() {
   System.out.println("Enter Destroy");
   System.out.println("Exit Destroy");
}

//synchronize the threads

synchronized public void run() {
   System.out.println("Enter RUN");
   ip = getCodeBase().getHost();
   System.out.println("IP Addr"+ip.toString());
   port = 5000;
   System.out.println("PortNumber="+port);
   String str = "";
   String rd;
   byte b;
   try{
        inSocket = new Socket(ip.toString(),port);
        is = new DataInputStream(inSocket.getInputStream());

        System.out.println("InStream is:>>" +is);
        while(true) {
            str = "";
            while((b = is.readByte()) != '&') { }
            while ((b = is.readByte()) != '\n') {
               str = str + (char)b; // read bytes, not Unicode characters
            }
            System.out.println("Input read is:>>" +str);
            updateBulbStatus(str);
         }
   }catch(Exception e) {
      System.out.println("An error occurred while reading the response.Try
                          reloading the page.");
   }
```

```
}

public void updateBulbStatus(String msg){
   System.out.println("Enter UpdateBulbStatus");
   System.out.println("Received String" +msg);

// BulbCanvas canvas = this.canvas;
      if (msg.equals("LED1on")) {
         redraw(1,Color.red);
      }
      else if (msg.equals("LED1off")){
         redraw(1,Color.green);
      }
      else if (msg.equals("LED2on")){
         redraw(2,Color.red);
      }
      else if (msg.equals("LED2off")){
         redraw(2,Color.green);
      }
      System.out.println("Exit updateBulbStatus");
}

//When the button is clicked, send  relevant information to control
//the LED on the board

public void actionPerformed(ActionEvent ev) {
   System.out.println("Enter Action Performed");
   String label = ev.getActionCommand();
   String s = new String();
   if(label.equals("D1")){
      s = "command=1&dummy=0";
      sendData(s);
   }else if(label.equals("D2")){
      s = "command=2&dummy=0";
      sendData(s);
   }
   System.out.println("Exit Action Performed");
}

//Send the data to the control program on the HTTP server

public void sendData(String s) {
   System.out.println("Enter SendData");
   String s1 = null;
   s1 =s;
   s1 = "GET /switch_cgi?" + s1 + " HTTP/1.1\r\n\r\n";
   System.out.println("Data to send =" + s1);
   try{
```

```
         System.out.println("rwSocket::"+rwSocket);

//       System.out.println("Received from Server:" +numbStr);
         Socket outSocket = new Socket(ipaddr, portNumber);
         os = new DataOutputStream(outSocket.getOutputStream());
         os.flush();
         os.write(s1.getBytes());
         outSocket.close();
   }catch(IOException _ex){
      System.err.println("Write exception"+_ex);
   }
   System.out.println("Exit Send Data");
   }
}
//End of file//
```